

EXERCISE 2

Basic trend analysis in EE



Introduction

Now that you have assembled a Landsat time series of seasonal composites, you are ready to analyze change through time. In this exercise, we will pick up where we left off with our script that generates cloud-free seasonal composites in a Landsat Time Series (LTS). We will use the Normalized Burn Ratio (NBR) to hone in on forest change and use a simple linear regression to look at trends through time. Finally, we will learn how we can create basic graphs in Earth Engine to provide insight into our data.

Objectives

- Learn how to compute and add a vegetation index to an image collection
- Execute a basic linear trend analysis on a Landsat time series and plot results

Required Data

- None – Everything we need is in Earth Engine!

Prerequisites

- **Completion of Exercise 1** (you can review code and pick up where we left off by accessing a copy of the final script from [the course repository](#)).
- **Google Chrome installed on your machine**
- **An approved Google Earth Engine account**



Table of Contents

Part 1: Add image transformation to the LTS script.....	3
Part 2: Visualize NBR through time on the Dixie NF.....	4
Part 3: Perform a simple linear trend analysis	6
Part 4: Review your final script	10



Part 1: Add image transformation to the LTS script

A. Return to the script you created in the last exercise

1. Open the script called **exploreLTS**.
 - i. If you skipped exercise 1, open the copy of the **Ex1.2_exploreLTS** script from [the course repository](#) and save it to your own account. Also, make sure to access and add the public repository built by the LandTrendr research group.
 - ii. To get read access to the eMapR repository containing the module we want (see below), visit the link below and accept the repository. This should add the `users/emaprlab/public` repository to your EE account.
 - iii. https://code.earthengine.google.com/?accept_repo=users/emaprlab/public

Note: The Environmental Monitoring, Analysis and Process Recognition (eMapR) lab at Oregon State University shares remote sensing tools and training materials through [their public LandTrendr Google Earth Engine \(LT-GEE\) GitHub repository](#). We will be using this module and EMAPR tools extensively in this course, and we recommend additional exploration of the eMapR LT-GEE guide.

2. Review your code so far. Your code should contain the following:
 - i. An imports section with a single geometry import representing the polygon area you used for exploring image processing in the last exercise.
 - ii. A single require statement for loading the LT-GEE module.
 - iii. A single call to the LT-GEE function to process and build an LTS with cloud, shadow, and snow mask.

B. Review the LT-GEE documentation on transformSRcollection

1. In your browser, navigate to the [LT-GEE module guide at this link](#) and review the description, parameters, result and example script for the **transformSRcollection** function.
 - i. We could easily accomplish the same task without the LT-GEE module a) using the **ee.Image.normalizedDifference** function to calculate NBR, and then using the map method to apply it across our collection. However, sometimes it is more convenient to use existing tools!

C. Use the transformSRcollection to add NBR to your LTS

1. Add the code below to your script, right after you create the collection with the variable name **'annualSRcollection'**.
 - i. Note that in this simple example, we are simply passing an array with one string element named 'NBR'.
 - ii. Add a print statement on the following line to explore the new collection in the console.

```
var indexCollection = ltgee
  .transformSRcollection(annualSRcollection, ["NBR", "NDVI", "TCW"])
print(indexCollection);
```

D. Explore the transformed LTS

1. In the console, expand the ImageCollection object and then expand the list of features.
2. You should see that the images in the ImageCollection each have additional bands for each image transformation specified (NBR, NDVI and Tasseled Cap Wetness), calculated from the multispectral image bands.

What are NBR, NDVI and TCW?

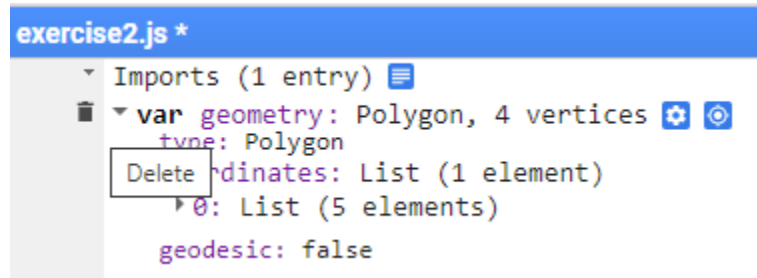
These are all image transformations that simplify Landsat image data, reduce noise and provide insight into vegetation. The Normalized Burn Ratio (NBR) and the Normalized Difference Vegetation (NDVI) are both indices based on simple band math. NBR is calculated from the near infrared (NIR) and short-wave infrared (SWIR) bands while NDVI is calculated from the NIR and Red bands. TCW stands for Tasseled Cap Wetness and is one of the important axes from the tried and true 'Kauth-Thomas' or 'Tasseled Cap' transformation. For more information on image transformation for extracting vegetation information, check out our [Introduction to Change Detection](#) course.

Part 2: Visualize NBR through time on the Dixie NF

Throughout the rest of this course, we will be exploring change on the Dixie National Forest near Brian Head, Utah. This area experienced significant forest mortality throughout the 1990s due to spruce beetle, and then experienced a significant wildfire in 2016.

A. Edit your geometry imports to examine a point on the Dixie NF

1. If you are using the code provided at the beginning of this exercise, skip ahead to section B—the point you need has already been added to the script. Still skim this section to review how to import points into your script!
2. Above your script, in the imports section, hover over the line that reads **var geometry: ...** until the trashcan icon appears.



3. Click the trashcan icon and when prompted, click Yes to confirm the deletion of your previous test study area.
4. Copy/paste the code below at the top of your script just above the module **require** statement.

```
var geometry = /* color: #d63000 */ ee.Geometry.Point([
  -112.74011959661296,
  37.75499572911073,
```

```
]);
```

5. After you have pasted this into your script, you can hover over the new text until it appears highlighted.
6. Next to the message stating: "geometry" can be converted to an import record. Click **Convert**

```
var geometry = /* color: #d63000 */ ee.Geometry.Point([-112.740119, 37.754996]);
// require t "geometry" can be converted to an import record. Convert Ignore
var ltgee =
```

7. You should now see the variable **geometry** added back to your import statements. Now it is a single point on the Dixie National Forest near Brian Head. Though we changed this variable to be point instead of a polygon, there is no need to update our scripts as we have used the same variable name and the functions we are using accept any type **ee.Geometry** object.

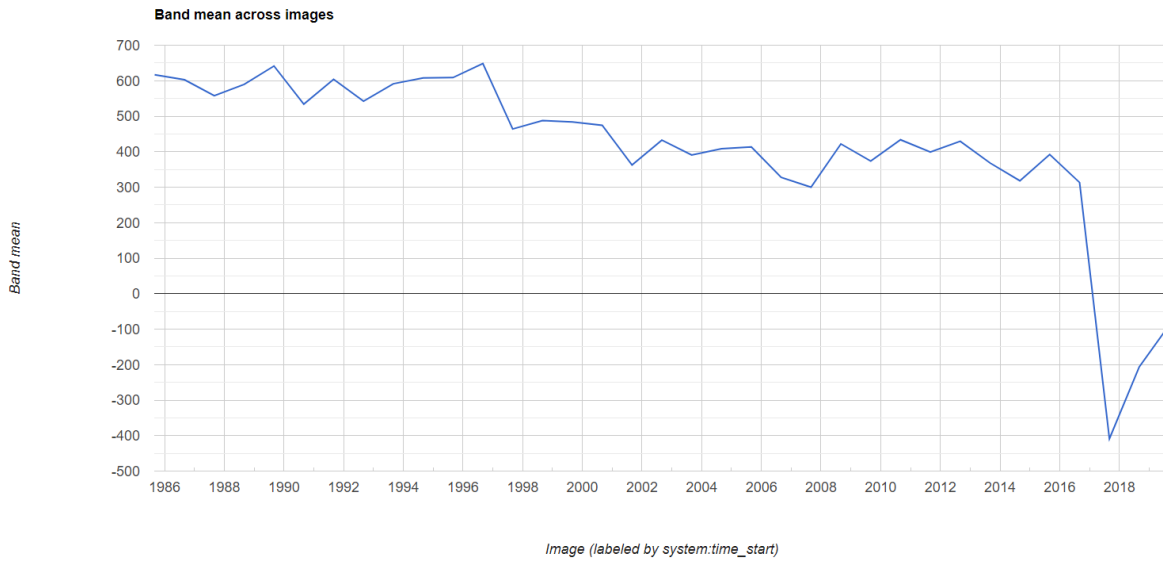
B. Add code to create a basic time series chart

1. Quickly review the user guide for an overview on creating charts in EE.
 - i. [Click this link to the EE chart documentation](#) and quickly skim the overview here, noting that you can create charts in the Code Editor console by passing data to the **ui.Chart** class and that there are several example scripts provided in the guide. We will be adapting the first example [in the documentation titled, Time Series Charts](#).
2. Look at the documentation to get the syntax for calling the **ui.Chart.series** function.
 - i. You should see that **ui.Chart.image.series** takes the parameters: imageCollection, region, reducer, scale, and X-axis coordinates. Only the first two parameters are required while the others have default values.
 - ii. Recall that reducers are how we aggregate data over time, space, image bands or other data structures in Earth Engine. Check the user guide for more [info on Reducers](#).
3. Add the code below to your script. Paste this snippet after the code to create and print **indexCollection**, but before your **Map.addLayer()** calls.

```
// Define a region of interest as a buffer around a point - Optional
var plotRegion = geometry.buffer(30);
// Center and zoom
Map.centerObject(geometry, 10)
// Create and print the chart - use NBR collection, buffered point, mean NBR
// values and 30-m scale for calculations
print(
  ui.Chart.image.series(
    indexCollection.select(["NBR"]),
    plotRegion,
    ee.Reducer.mean(),
    30
  )
)
```

```
);
```

4. Save your script and click **Run** to observe the result in the console.
 - i. You should see a simple spectral trajectory graph, like the one below, appear in the console.



Note: This trajectory suggests a fairly stable condition from 1985 to 1997 and then a slow, gradual decrease in NBR until about 2007. NBR values then appear to stabilize until 2016 and then a significant disturbance causing a drastic drop in NBR in 2017.

This trajectory makes sense given this area experienced an outbreak of spruce beetle that causes a relatively slow die off of the older, larger trees. These standing dead trees later provided ample fuel for the 2017 Brian Head fire.

Part 3: Perform a simple linear trend analysis

One of the simplest ways to analyze a Landsat Time Series would be to simply fit linear trend lines through time. If stable conditions through time appear as flat line trajectories, and loss of vegetation appears as a downward sloping line, we could simply calculate the slope of NBR through time and identify the negative slopes to find areas that have experienced vegetation loss.

A. Review the Linear Regression reducers in the User Guide

1. Open the [user guide section on Linear Regression](#) and quickly review to get the gist, noting the following key points:
 - i. There are a few different ways to perform regression analysis. All are **ee.Reducers**.
 - ii. For **ee.ImageCollection** objects, we can use the **linearFit** or **linearRegression** reducers.

2. Look at [user guide example demonstrating linearFit](#). (You can inspect it in your browser, or feel free to copy to a new script and explore! Just make sure to save your existing script before opening a new one with the linearFit example.) Note the following points:
 - i. The first line in the example “adds a time band to the image.” That is, it creates a new band where the values represent the time at which the image was collected. In order to calculate change over time across an image using Google Earth Engine methods, we need to create this band.
 - ii. The example creates an object called **linearFit** object. To create the object, the command first a) selects two bands from the input image collection—first, the predictor or x variable, which in this case is the newly-created time band. The second band is the response or y variable, which in the example is called ‘**pr_mean**’, or mean precipitation. Second, b) the two input bands are passed to a [reduce](#) command that applies the **ee.Reducer.linearFit** to calculate the linear relationship between the predictor variable and the response variable.
 - iii. Reflect: The **linearFit** method is a **reducer**. What do you think that means?
 - iv. The result of this command is a single image with two bands: **scale**, representing the slope of the regression, and **offset**, representing the intercept of the regression. Clear as mud? Let’s give it a try!

B. Add a time band to the NBR time series

1. Copy the **createTimeBand** function directly from the user guide and add it to the top of your script, just below the **require** statement that imports the **ltgee** module.

```
// This function adds a time band to the image.
var createTimeBand = function (image) {
  // Scale milliseconds by a large constant to avoid very small slopes
  // in the linear regression output.
  return image.addBands(image.metadata("system:time_start").divide(1e18));
};
```

C. Use the map method to apply the function across the NBR collection

1. Locate the line in your script where you declare the variable **indexCollection** and add (called chaining) the **map** method at the end. You will chain, or add, a **.map** function which takes the newly declared **createTimeBand** function as an argument.
 - i. Your code should look like this:

```
var indexCollection = ltgee
  .transformSRcollection(annualSRcollection, ["NBR", "NDVI", "TCW"])
  .map(createTimeBand);
print(indexCollection);
```

2. Click run again to see the effect of adding this

- i. In the console, expand the ImageCollection object to examine the bands. The ImageCollection stores each image, and each band of each image, within a list of **features**. Navigate through ImageCollection > features > 0 > bands. When you expand the band list of one of the images in this collection, you will see a fourth band with the name "system:time_start".

```

▼ ImageCollection (35 elements)
  type: ImageCollection
  bands: []
  ▼ features: List (35 elements)
    ▼ 0: Image (4 bands)
      type: Image
      ▼ bands: List (4 elements)
        ▶ 0: "NBR", float ∈ [-1000, 1000], EPSG:4326
        ▶ 1: "NDVI", float ∈ [-1000, 1000], EPSG:4326
        ▶ 2: "TCW", double, EPSG:4326
        ▶ 3: "system:time_start", double, EPSG:4326
      ▶ properties: Object (2 properties)
  
```

D. Use the linearFit() reducer to fit a regression for NBR vs. time

1. Add the code below to your script after you create the **indexCollection** with the time band added.

- i. Note that we are chaining methods again, first using the **.select** method to set up our x and y variables for the regression and then chaining the **.reduce** method with the **linearFit** reducer as an argument

```

var linearFit = indexCollection
  .select(["system:time_start", "NBR"])
  .reduce(ee.Reducer.linearFit());

```

2. Use a print statement to explore the newly created linearFit object.

- i. Add the code below and click run again.

```

print(linearFit) // a two-band image with 'scale', or slope, and 'offset', or
intercept

```

Note: While this approach could be powerful and insightful for picking up some kinds of change (e.g., gradual constant change), it has significant shortcomings that are readily apparent from looking at the trajectory for the example point on the Dixie NF. While a simple line might fit the first part of the trajectory, this trajectory appears to have at least 3 distinct 'periods'. It would be impossible to fit the entire trajectory with a single line and a more sophisticated approach is warranted.

E. Add the linearFit image to the map

1. Use the **Map.addLayer** method to add the image to the map. Here, we are visualizing **scale**, or the slope of the linear regression between NBR and time.

```

Map.addLayer(linearFit.select("scale"), {}, "NBR linear fit slope");

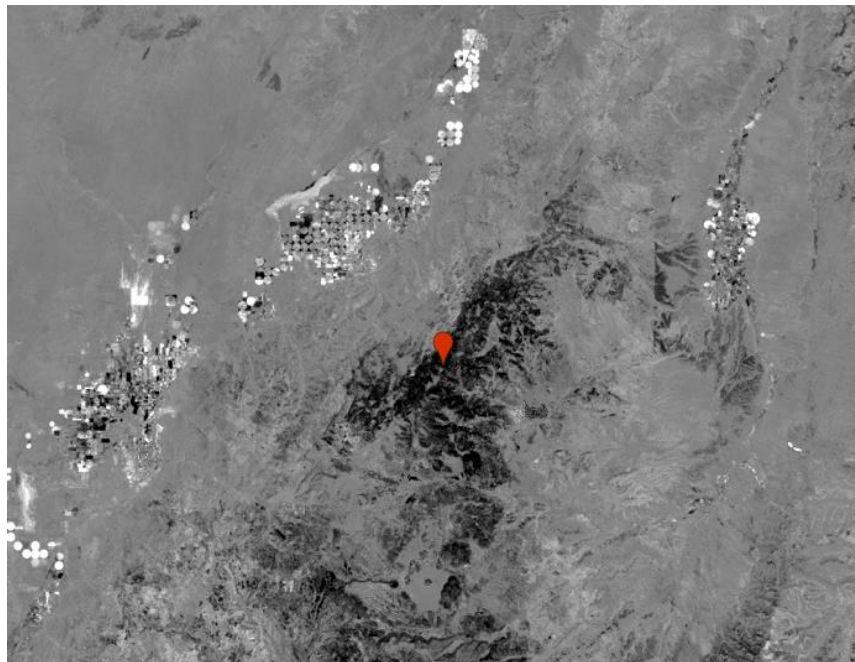
```


2. Click Run to see the result. The image is binary because of the large number rescale in the image transformation function. Add a stretch to improve the visualization.
 - i. Use the inspector to click around and get some values from the image to improve the visualization. You will find very large positive and negative integers and get a sense of the range to create a stretch.
 - ii. Replace the code above with the code below to adjust the visualization parameters for the linear fit image.

```
Map.addLayer(
  linearFit.select("scale"),
  { min: -5e8, max: 5e8 },
  "NBR linear fit slope"
);
```

F. Explore the NBR slope image

1. Use the map tools to zoom and pan around in the NBR slope image. Use the inspector to explore the image slope values.
 - i. You can use the map vector tools to move the geometry point and rerun the script to observe spectral trajectories in different areas.



Note on interpreting this image:

Negative slopes appear dark and indicate areas where vegetation loss has occurred while positive slopes appear bright and indicated areas where vegetation has increased. From this, you can easily the perimeter of the 2017 Brian Head fire as well as new pivot irrigation.

Part 4: Review your final script

A. Read through and check your understanding

1. Make sure you understand what each line of code is doing and feel free to add notes for your future self or others with whom you may share your script.

B. Reformat for clarity

1. The first part of your script sets up your analytical environment. This part should contain the `geometry import`, then the `require` statement to load the `ltgee` module, followed by the function that we created to add time bands.
2. The next part of your script contains your image processing, and later adds your results to the map.
3. Your final script should resemble the screenshot below.
 - i. You may compare it to the final script found in [the course repository](#).



```
Imports (1 entry)
var geometry: Point (-112.74, 37.75)

1  /// SETUP
2
3  // import LandTrendr
4  var ltgee = require("users/emaprlab/public/Modules/LandTrendr.js");
5
6  // This function adds a time band to an image.
7  var createTimeBand = function (image) {
8    // Scale milliseconds by a large constant to avoid very small slopes
9    // in the linear regression output.
10   return image.addBands(image.metadata("system:time_start").divide(1e18));
11 };
12
13 /// IMAGE PROCESSING
14
15 // create Landsat time series (LandTrendr)
16 // composited, masked, time series of images with additional processing for historic Landsat data
17 var annualSRcollection = ltgee.buildSRcollection(
18   1985,
19   2019,
20   "06-20",
21   "09-20",
22   geometry,
23   ["cloud", "shadow", "snow"]
24 );
25
26 // add to map
27 Map.addLayer(
28   annualSRcollection,
29   { bands: ["B4", "B3", "B2"], min: 250, max: 3000, gamma: 1.6 },
30   "cloud/shadow/snow masked SR collection"
31 );
32
33 // create transformed LTS
34 // calculate NBR, NDVI, TCW and add a time band
35 var indexCollection = ltgee
36   .transformSRcollection(annualSRcollection, ["NBR", "NDVI", "TCW"])
37   .map(createTimeBand);
38 print(indexCollection);
39
40 /// CONDUCT A SIMPLE TREND ANALYSIS
41
42 // fit a trend line
43 // regression: NBR ~ time
44 var linearFit = indexCollection
45   .select(["system:time_start", "NBR"])
46   .reduce(ee.Reducer.linearFit());
47
48 // view trend line
49 print(linearFit) // a two-band image with 'scale', or slope, and 'offset', or intercept
50
51 /// VISUALIZE
52
53 // Define a region of interest as a buffer around a point - Optional
54 var plotRegion = geometry.buffer(30);
55
56 // Create and print the chart - use NBR collection, buffered point, mean NBR values and 30-m scale for calculations
57 print(
58   ui.Chart.image.series(
59     indexCollection.select(["NBR"]),
60     plotRegion,
61     ee.Reducer.mean(),
62     30
63   )
64 );
65
66 // add linear fit to map with visualization stretched to represent values
67 Map.addLayer(
68   linearFit.select("scale"),
69   { min: -5e8, max: 5e8 },
70   "NBR linear fit slope"
71 );
```

This is a basic example of how to follow standard conventions and patterns for writing code. Especially as your code becomes more complex, writing code that is well-annotated (has lots of comments) and follows a standard order helps other users read and understand your code—and can help future you understand your code better, too! Following this basic order of commands is a good place to start. You can gain other guidance for organization by perusing other users' GEE code. For more advanced users, GEE supplies a list of [coding best practices](#) to make your operations run more efficiently.





C. Save, close and get ready to move on to the next section!

Congratulations! You have completed this exercise and explored several new tools and methods for working with image time series data in Earth Engine. You have learned how to conduct a very simple trend analysis in just a few lines of code. In the next few exercises, you will build on this knowledge and learn how to use more robust tools for identifying and mapping disturbance.

